An integrated object-oriented approach for parallel CFD

Dominique Eyheramendy, David Loureiro, Fabienne Oudin-Dardun

An integrated object-oriented approach for parallel CFD

Dominique Eyheramendy[1],David Loureiro[2], Fabienne Oudin-Dardun[3]

[1] Ecole Centrale Marseille LMA-CNRS, Technopôle de Château-Gombert 38 rue Frédéric Joliot Curie, 13451 Marseille, France
dominique.eyheramendy@ec-marseille.fr
[2] LIP/ENS-LYON/INRIA/CNRS/UCBL, 46 Allée d'Italie, 69364 Lyon, France
david.loureiro@ens-lyon.fr
[3] Université de Lyon, Université Lyon 1, CNRS UMR5208, Institut Camille Jordan, CDCSP-ISTIL, 15 Blvd Latarget, 69622 Villeurbanne, France
fabienne.oudin@univ-lyon1.fr

Abstract. In this paper, we present a global computer science approach to deal with parallel computations. The proposed approach consists in managing at the same level either multithreading or distributed strategies, whatever the computation may be. The integration of the concept is held in a Java framework which proposes both, a pure object-oriented paradigm and, convenient libraries to deal with threads management and communications schemes. The approach is illustrated on a domain decomposition method for a Navier-Stokes flow.
Keywords: Finite elements, object-oriented programming, domain decomposition, multi-threaded computing, distributed computing.

1 Introduction

In this paper, we present the key ideas of a Java application to a finite element code. This approach is based on the reusability of code and the portability of application in the case of parallel application. The key idea of the developments is that the parallel algorithms are programmed within the application using a single programming concept and a single language. In classical approaches, the programming language (C, C++, Fortran) is associated to additional libraries such as PVM or MPI. Existing libraries in the Java environment are used here. Thus, the maintenance of the application is made easier, which is important from an industrial point of view. The proposed approach offers the advantage of a high level of reusability of the different parts of the code whatever the computational strategy is. The consequence of it is a high reliability of the software for domain decomposition methods. This reliability is obtained through a classical object-oriented paradigm which allows the programmer

to separate the management of finite elements data, the solution algorithms, the management of different processes and the communication schemes.

In section 2, we give a state of the art for Java applications in Computational Mechanics that most often includes parallel computations. In section 3, we described two different implementations of an overlapping Schwarz domain decomposition method within the same finite element code. In section 4, we give a numerical application that has been run using the same code on different systems.

2 Java in Computational Mechanics

Until now, most of the developments in Java computational mechanics have been considered by the computational science community, in general concerned by parallel computations. In the domain of numerical computations, Java retained some attention for its networking capabilities and its Internet easy portability. E.g. in [1], a trivial application based on a boundary element method has been developed. Similar developments may be found directly on the Internet, including basic finite element applications, such as in [2] for an application of fracture mechanics. In the computational mechanics community, Java is often considered as a simple tool to produce applications on the Internet and/or to effectuate computation on the network. For example, Java kind technologies are often used to couple and manage traditional codes written in C/C++/Fortran. This permits the developers to use ancient codes or part of code in coupled applications. A consequence of it is to keep a real computational efficiency. E.g. in [3], an interactive finite element application based on a coupled C++/Java is described. Comparative tests with FORTRAN and C are conducted on small problems using direct solvers based on tensor computations; this aims at illustrating the high efficiency computational potential of Java. In [4], the development of GUIs is put in prominent position on an unstructured mesh generator. In [5], the Java code CartaBlanca is presented. It is an environment for distributed computations of complex multiphase flows. Based on a finite volume approach, a solution scheme based on a Newton-Krylov algorithm is described. The code exhibits good performances. A similar environment has been developed to simulate electromagnetisms problems in [6]. Both applications show the high potential of the approach to design more complex and general computational tools in mechanics for example. These developments exhibit the networking facilities provided in Java. A large number of publications shows the interest of Java and its efficiency: direct solution of linear systems [7], FFT and iterative and direct linear systems solvers on Euler type flows [8], solution of Navier-Stokes flows [9] and [10]. Again, in all these papers, multiprocesses' management and networking capabilities of Java are put forward. In [8], [9] and [11], performances of Java are tested on simple matrix/vector products. Compared to C/C++ code, only 20 to 30% of efficiency is lost. More recently, the description of a finite element code in Java was proposed in [12]. The proposed design remains rather similar to the existing ones based on classical object-oriented approaches. Nevertheless, this paper shows that it is possible to develop a global code based on a pure Java approach. The latter can be applied to design large complex

applications in computational mechanics taking into account complexity in modern computational mechanics: multiscale, multiphysics and multiprocesses applications. Firstly, enhanced and homogeneous data organization schemes to deal with complexity are proposed in Java. Secondly, developing a global application in a uniform environment including all the libraries needed is somehow an attractive idea for scientists and industrials. It is worth noting that similar approaches exist in C# (e.g. see [15], [16]). This brief analysis shows that even if performances do not achieve the one of C/C++/Fortran but get close to a certain extent, Java or similar approaches can be used in computational mechanics including CFD. These strategies offer the main advantage to design the simulation tools such a way that the technical strategies and the numerical algorithms are integrated in a seamless way.

3 Schwarz multiplicative multithreaded and distributed applications

The Schwarz multiplicative algorithm is described in [11]. In Fig. 1, the same decomposed domain is shown to be solved in a multi-threaded application or in a distributed application. Solving both problems is made through the same basic code for the physics and numerical algorithm. A description of the code goes beyond the scope of this paper and can be found in [13], [17] and [18]. The description of the multithreaded application is given in [17]. As shown in Fig. 1, the general structure of the code organization is the same. In the distributed version, the global Schwarz algorithm is managed through a thread located on the computer 1. The algorithm lies in two points: the management of the domains located on alternative computers and the communication schemes between these domains located on various computers. The global management is held by the way of distributed objects. The Java RMI package is used for that purpose. It permits the programmer to keep a natural and homogeneous organization of classes to deal with the domain decomposition algorithm. The consequence of it is to keep exactly the same structure as in both algorithms, managing real objects in the one case, and distributed objects in the other case. The later remains seamless for the programmer. The Schwarz algorithm is given in both cases as follows (programming details are omitted in the following code):

```
public void solve( )
{
    // . . . INITIALIZING OF THE SOLVER
    for(int it = 0; (it < maxIteration) && (! iteration.converged() ); it++)
    {
        for( int color = 0 ; color < 2 ; color++ )
        {
            int anteColor = ( color + 1 ) % 2 ;
            //***************** solution *****************
            // SOLUTION BLOCK OMMITED
            //*************** exchanges ***************
            // EXCHANGES BLOCK OMMITED
```

```
            //****************************************
        }
        // FINALIZING THE ITERATIONS
        // . . .
    }
}
```

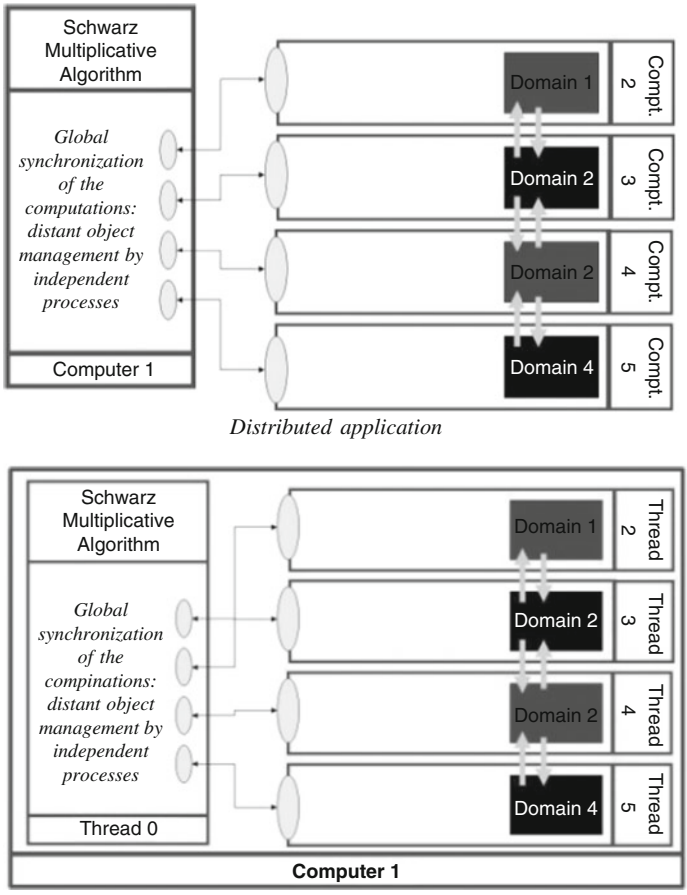


Fig. 1. Multi-threaded application versus distributed application.

The major differences between both approaches remain the initialization phases in the different threads (solution, exchanges,. . .). We give here the example of the thread creation to solve the physical problem for a given domain:

```
Thread threadii = new Thread ( new Runnable ()
    {
```

```
        int number = Ni ;
        public void run()
        {
            domains[number].solveSchwarz () ;
        }
    } ) ;
```

The variable called domains[number] represents the domain (numbered number) to be solved at a given time. In the distributed version, the same piece of code becomes:

```
Thread threadii = new Thread( new Runnable()
    {
        public void run() {
            try {
                ((ServerDomain) listOfServerDomain.get(indice0a)).solveSchwarz();
            } catch (RemoteException e) { }
        }
    });
```

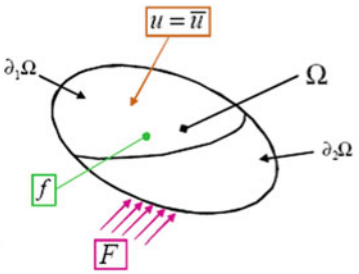
In this piece of code, the equivalent domain is called using the code: listOfServerDomain.get(indice0a) but the global scheme remains the same. In the first approach, the domain is located in the same memory space, as in the second approach, it is located on a separate computer. The lack of efficiency in communication schemes is the main drawback of the use of RMI distributed objects. The latter is not a problem for the management of the global Schwarz algorithm but produces a bottleneck when exchanging data. This is the reason why communications are programmed using the Socket Java class which a point to point communication scheme. This scheme is implemented at the level of boundary conditions object where the information provided by the neighbor domain id needed. Once again, only a local modification is needed within the global framework. For illustration purposes, both schemes are applied to a Navier-Stokes flow. For programming details in Java, the reader may refer to [19].

4 Numerical application

To illustrate both approaches, we study a Galerkin Navier-Stokes formulation stabilized by adding least-squares type terms. The equations of the problem and the formulation are presented in Fig. 2. Linearization of the problem is introduced in a Newton like scheme in the Schwarz multiplicative scheme. A direct linear system solver based on a Crout decomposition is used to solve the linear system at each iteration. Note, the same code is compiled once and run on all systems. The multi-threaded version of code is run on a SGI Altix 16-processors Itanium2 1.3Ghz, 32 Go of RAM. The distributed version is run on a set of simple cluster of PC linked by a classical network. The code used for both applications is exactly the same. The global solution algorithm and the boundary conditions in charge of the communication scheme are of course not the same. The formulation is applied to the computation of a flow through a set of cylinders shown in Fig. 3. The domain is a periodic layer of cylinders. Numerical results are given in Fig. 3. First, the pressure contour on a

typical cell is plotted. Secondly, the mean velocity computed overall the domain is given with respect to the gradient of pressure over the cell in the direction of the main flow. The latter represent the homogenized flow through the cylinders. For low velocities, the relation between the mean velocity and the gradient of pressure is linear. From a global point of view this can be assimilated to a Darcy's flow. But as far as the mean velocity is increasing the linearity disappears. The advection term is no more negligible and the global flow cannot be assimilated to Darcy's flow. It shows the influence of the advection term on the homogenized flow, that cannot any more considered as a linear Darcy's flow. Filtration laws could be established such a way. The same computation has been held using both approaches. From a practical point of view, we show that the same code can be run on heterogeneous systems. As both systems are different, we cannot compare the efficiency of both algorithms. This goes beyond the scope of our qualitative test. We show here that we can easily switch from a computer to another one without any problem using exactly the same code and without compiling it. This feature can be very interesting from an industrial point of view.

$$
\begin{aligned}
&\sigma_{ij,j} + f_i = \rho(u_i u_{i,j} + u_{i,t}) && \text{on } \Omega\times T\\
&u_{i,j} = 0 && \text{on } \Omega\times T\\
&\sigma_{ij} n_j = F_i && \text{on } \partial_2\Omega\times T\\
&u_i = u_i && \text{on } \partial_1\Omega\times T\\
&\sigma_{ij} = -p\delta_{ij} + 2\mu\varepsilon_{ij}(u) && \text{on } \Omega\times T\\
&\varepsilon_{ij}(u) = \frac{1}{2}(u_{i,j} + u_{j,i}) && \text{on } \Omega\times T\\
&u_i(0) = u_{i0} && \text{on } \Omega \text{ at } t = 0
\end{aligned}
$$



• Given f, find $(u^h, p^h) \subset ((\mathcal{S}^h)_u \times (\mathcal{P}^h)_u)$ such that for each $(w^h, q^h) \subset ((\mathcal{W}^h)_u \times (\mathcal{P}^h)_u)$, one has :

$$
\int_\Omega \rho u_j^h u_{i,j}^h + \rho u_{i,t}^h dv - \int_\Omega 2\mu\varepsilon_{ij}(u^h)\varepsilon_{ij}(w^h)dv + \int_\Omega p_h w_{i,i}^h dv + \int_\Omega u_{i,i}^h q^h dv - \int_\Omega f_i w_i^h dv
$$

$$
+ \sum_{\Omega^e \in \Omega^h} \left[\int_{\Omega^e} (\rho u_j^h u_{i,j}^h + \rho u_{i,t}^h - 2\mu\varepsilon_{ij,j}(u^h) + p_{,i}^h - f_i)\tau_{mom}(\rho u_j^h w_{i,j}^h + q_{,i}^h)dv \right] = 0
$$

Fig. 2. Navier-Stokes model. Initial-boundary value problem and stabilized finite elements formulation.

5 Conclusion

In this paper, we have presented two computational approaches based on the same numerical algorithm, a Schwarz overlapping domain decomposition method. The

first approach is a multithreaded approach of the algorithm; the second one is a distributed version of it. In the first one, which is to be run on a share memory system, no communication between the domains is needed. The second one is based on two different communication schemes. The first one permits a master to manage the global Schwarz algorithm based on a classical object-oriented approach for distributed objects. Communications for data exchanges are implemented using classical sockets. This point to point communication scheme allows us to achieve efficiency. The implementation is held in Java which ensures the code to be run directly on all the systems. The same code is run for both applications. The advantages of such an approach are:

1. from a programmer point of view: a single language and a single approach for all the algorithms which means simplicity and reliability due to the fact that the finite element core remains identical.
2. From a user point of view: the same code can be used on heterogeneous systems depending on the availability of different systems.

We advocate that such an approach may simplify a lot the use of complex systems for single applications. This opens new tracks in the design of code that can be used in the context of either a shared memory system or a distributed memory system. In this context, mixing both approaches within a single application, taking advantage of the heterogeneous computers systems available at a given time is made possible in a simple way.

[1] M. Nuggehally, Y.J. Lui, S.B. Chaudhari and P. Thampi, An internet-based computing plateform for the boundary element method, Adv. In Engrg. Software, 34 (2003) 261-269.

[2] G.P. Nikishkov and H. Kanda, The development of a Java engineering application for higher-order asymptotic analysis of crack-tip fields, Advances in Engineering Software 30 (1999) 469-477.

[3] G.R. Miller, P. Arduino, J. Jang and C. Choi, Localized tensor-based solvers for interactive finite element applications using C++ and Java, Comp. & Struct. 81 (2003) 423-437.

[4] R. Marchand, M. Charbonneau-Lefort and M. Dumberry, ARANEA, A program for generating unstructured triangular meshes with a Java Graphics User interface, Comp. Phys. Communications, 139 (2001) 172-185.

[5] N.T. Padial-Collins, W.B. VanderHeyden, D.Z. Zhang, E.D. Dendy and D. Livescu, Parallel operation of CartaBlanca on shared and distributed memory computers, Concurrency and Computation: Practice and Experience 16 (2004) 61-77.

[6] L. Baduel, F. Baude, D. Caromel, C. Delb, N. Gama, S. El Kasmi and S. Lanteri, A parallel object-oriented application for 3-D electromagnetism, ECCOMAS 2004, Jyvskyl, Finland (2004).

[7] G.P. Nikishkov, Y.G Nikishkov and V.V Savchenko, Comparison of C and Java performance in finite element computations, Computer & Structures, 81 (2003) 2401-2408.

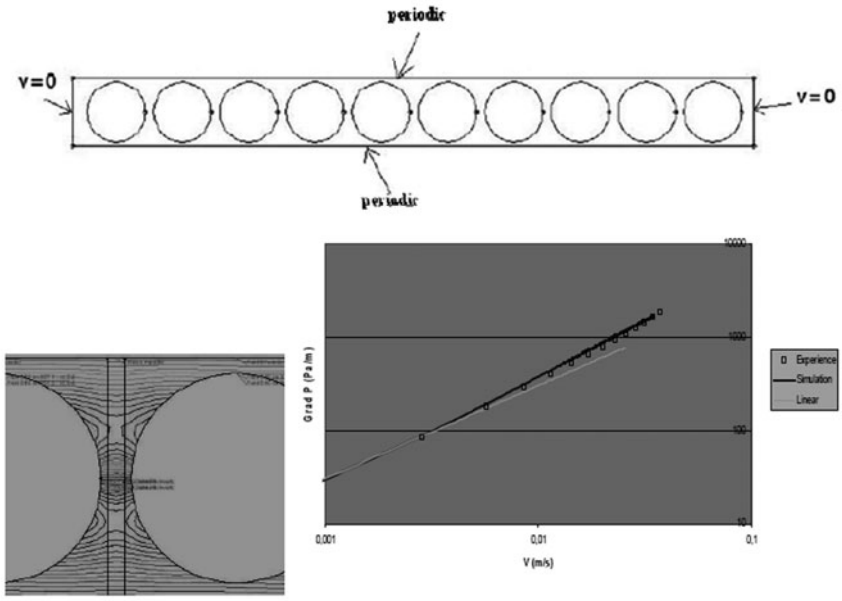


Fig. 3. Flow through cylinders. Computational domain, decomposed domain, numerical results: detail of iso-pressures, curve velocity/gradient of pressure (Experiment and simulation).

[8] J.M. Bull, L. A. Schmith, L. Pottage and R. Freeman, Benchmarking Java against C and Fortran for Scientific Applications, Joint ACM JavaGrande - ISCOPE 2001 Conference, Stanford Universtity, June 2-4, 2001.

[9] J. Huser, T. Ludewig, R.D. Williams, R. Winkelmann, T. Gollnick, S. Brunett and J. Muylaert, A test suite for high-performance parallel Java, Advances in Engineering Software, 31 (2000) 687-696.

[10] C.J. Riley, S. Chatterjee and R. Biswas, High-performance Java codes for computational fluid dynamics, Concurrency and Computation: Practice and Experience 15 (2003) 395-415.

[11] D. Eyheramendy, Object-oriented parallel CFD with JAVA, 15th International Conference on Parallel Computational Fluid Dynamics, Eds. Chetverushkin, Ecer, Satofuka, Priaux, Fox, Ed. Elsevier, (2003) pp. 409-416.

[12] D. Eyheramendy, Advanced object models for mathematical consistency enforcement in scientific computing, WSEAS Transactions on Mathematics, vol. 4, N° 4, (2005), pp. 457-463.

[13] D. Eyheramendy, High abstraction level frameworks for the next decade in computational mechanics, Innovation in Engineering Computational Technology, Eds. B.H.V. Topping, G. Montero and R. Montenegro, ©Saxe-Cobourg Publications, Chap. 3, (2006) pp. 41-61.

[14] G.P.Nikishkov, Object oriented design of a finite element code in Java. Computer Modeling in Engineering and Sciences 11 (2006) pp. 81-90.
[15] R.I. Mackie, Object-oriented design of pre-conditionned iterative equation solvers using .NET, Proceedings of 5th Int. Conf. on Engineering Computational Technology, Las Palmas de Gran Canaria, Spain, 12-15 Sept. 2007.
[16] R.I. Mackie, Lessons learnt from using .NET for distributed finite element analysis, Proceedings of 11th Int. Conf. on Civil, Structural and Environmental Engineering Computing, St. Julians, Malta, 18-21 Sept. 2007.
[17] D. Eyheramendy and F. Oudin, Advanced object-oriented techniques for coupled multiphysics, In Civil Engineering Computation: Tools and Techniques, Ed. B.H.V. Topping, ©Saxe-Cobourg Publications, ISBN 978-1-874672-32-6 Chap. 3, (2007) pp. 37-60.
[18] D. Eyheramendy, Advanced object models for mathematical consistency enforcement in scientific computing, WSEAS Transactions on Mathematics, vol. 4, N° 4, (2005), pp. 457-463.
[19] D. Flanagan, Java in a Nutshell, Fourth edition, Ed. O'reilly (2002).